REMARKS

By this amendment, claims 1-6, 8-15, and 17-23 are pending, in which claims 7 and 16 are canceled without prejudice or disclaimer, claims 1, 5, 10, and 14 are currently amended, and claims 19-23 are newly presented. No new matter is introduced.

The Office Action mailed July 16, 2003 objected to claims 5 and 14 for informalities and rejected claims 1-18 under 35 U.S.C. § 102 as anticipated by *Biliris et al.* (US 5,590,327).

In response to the objection, claims 5 and 14 have been amended as helpfully suggested by the Examiner.

The rejection of claims 1-6 and 10-15 is respectfully traversed because *Biliris et al.* does not disclose the limitations of claims 1-6 and 10-15. For example, independent claims 1 and 10, as amended to incorporate the subject matter of former claims 7 and 16 respectively, recite: "generating a layout for the object in a high-order language based on the definition of the object and the size and alignment of the one or more primitive types." The generation of object layouts based on the size and alignment advantageously results in an object system for use in a run-time environment that is both portable and maintainable (see Spec. p. 7).

This feature, however, is not shown in *Biliris et al.* Rather, *Biliris et al.* is directed to a "method for making data objects having hidden pointers persistent" (abstract), such as C++ objects that use hidden pointers for virtual functions and virtual base classes (col. 2:13). The specific problem addressed by *Biliris et al.* is that the values of the hidden pointers may vary across invocations of a program and so may not valid if retrieved from a persistent storage, but there is not direct mechanism in C++ for the user to fix these pointers since they are hidden (col. 2:46-57). Accordingly, *Biliris et al.* describes in FIG. **8** a method for retrieving an object from persistent storage (step **71**) and reinitializing the retrieved object (step **72**) in such a way that that

only the hidden pointer initialization (step **63**) is made, skipping the memory allocation (step **61**) and data member initialization (step **62**).

Biliris et al., however, does not disclose 'accessing a platform-specific description of size and alignment of the one or more primitive types" or" generating a layout for an object in a high-order language based on ... the size and alignment of the one or more primitive types." In fact, *Biliris et al.* is unconcerned with either the problems with or the details of generating object layouts, and the portions cited in *Biliris et al.* do not go beyond the ordinary use of C++. For example, col. 4:30-50 shows a class definition including an array defined as char firstname[MAX] for which the task of the C++ compiler is to "generate code" (col. 1:61-63). The MAX parameter, however, is not an "alignment" as recited in claims 1 and 10, and "code" is too generic of a term to disclose the specifically recited "high-order language."

In fact, *Biliris et al.* fails to have any disclosure of alignment, and the portion cited in the Office Action with respect to former claim 7, col. 11:11-15, merely discusses an overloaded definition of the new operator(size_t, _ode *p) that takes and never uses a size_t parameter and a pointer cast to a dummy class _ode *. There is no mention, however, of a "description of size an alignment" of a type, nor generating an object layout "based on the size and alignment." In fact, compilation of the overloaded operator new() in col. 11 does not result in "generating a layout for the object in a high-order language."

As for the rejection of claim 8-9 and 17-18, that too is respectively traversed. Claims 8 and 9 recite "generating a plurality of layouts, corresponding respectively to the incompatible platforms," but *Biliris et al.* does not even disclose multiple platforms, not to mention whether the platforms are incompatible. FIG. 3, referenced in the Office Action, shows a memory layout for a student object **31** that includes a student subobject **58** and a person subobject **36**. However, there is no disclosure that the memory space for student object **31** exists on a plurality of

incompatible platforms. It is unclear what understanding of "incompatible platform" exists that is both consistent with its plain meaning to one of skill in the art and the claims (e.g. "platform-specific compiler" in claim 9) and supportive of the rejection.

Newly presented claims 19-23 are patentable over *Biliris et al.* because *Biliris et al.* does not show the recited "padding element." No new matter is added by the recital of "padding element" since adequate descriptive support exists throughout the specification, including page 17, second paragraph.

Therefore, the present application, as amended, overcomes the objections and rejections of record and is in condition for allowance. Favorable consideration is respectfully requested. If any unresolved issues remain, it is respectfully requested that the Examiner telephone the undersigned attorney at 703-425-8516 so that such issues may be resolved as expeditiously as possible.

Respectfully Submitted,

DITTHAVONG & CARLSON, P.C.

10/16/03
Date

Stephen C. Carlson
Attorney/Agent for Applicant(s)
Reg. No. 39929

10507 Braddock Rd
Suite A
Fairfax, VA 22032
Tel. 703-425-8516
Fax. 703-425-8518